INTRODUCTION to DEM software LIGGGHTS

GONCALVES DE O C, Joao,

LECROISEY Solène

September 2020

Contents

1 Introduction **2**

2 Basic Informations **2**
2.1 Command lines . . . 2
2.1.1 Example 1 . . . 2
2.1.2 Exemple 2 . . . 3
2.2 Script Structure . . . 3
2.2.1 Initialization . . . 3
2.2.2 Détails d'exécution . . . 4
2.2.3 Model . . . 4
2.2.4 Exécution . . . 5

3 Avanced Commands **6**
3.1 Variable . . . 6
3.2 Jump and Label . . . 7
3.3 Exporting Data . . . 7
3.4 write_data & read_data . . . 8
3.5 Shell . . . 8
3.6 Print . . . 8

4 Installing LIGGGTHS **8**

5 OpenSCAD **9**

Nomenclature **10**

1 Introduction

This document aims to give an idea of how to make a script for the LIGGGHTS program. The latter is an Open Source particle simulation software using the Discrete Element Method (DEM - Discrete Element Method), developed mainly by Christoph Kloss and distributed by DCS Computing GmbH.

LIGGGHTS is based on the LAMMPS software, a classical molecular dynamics simulator, with the aim of improving the calculations of granular materials for industrial applications.

This document is divided into several parts. First, some basic examples of the LIGGGHTs language will be exemplified. Next, the different parts of the simulation initialization will be described. After that, several advanced commands will be explained. Finally, a point in the software installation and the integration with OpenScad will be mentioned.

All the commands described in this text are also explained in the software documentation. We strongly encourage anyone willing to learn the software to read them. It is easily accessible from the CFDEM website.

2 Basic Informations

2.1 Command lines

LIGGGHTS is a program coded in the C++ language, but to create a working simulation it is necessary to use a specific language defined only for this program. In most cases, a line of code is written in the following form:

mainWord **ID** targetGroupID style **args** keyWords **keyArgs**

- mainWord : is the main word. It describes the class of modification that will be apported;
- **ID** : is the chosen identification name of the command ;
- targetGroupID : is the name of the group of particles that will be modified by the command line (not always relevant);
- style : defines the style of mainWord, to be executed. ;
- **args** : necessary values for the chosen style ;
- keyWords : optional modification of the chosen style;
- **keyArgs** : necessary values for the chosen keyWords;

2.1.1 Example 1

Adding gravity to the model

fix **grav** all gravity **9.81 vector 0.0 0.0 -1.0**

- The chosen mainWord is of the type 'fix';
- **'grav'** is the identification name chosen for the given fix ;
- Gravity will affect all particles of the simulation, thus the targetGroupID is 'all';
- 'gravity' is the type of 'fix' being called;
- The chosen 'gravity' type requires a multiplication factor and a vector (check LIGGGHTS documentation for more information). In this example, the multiplication factor chosen is **9.81**, and the vector **(0,0,-1)**. Thus indicating an acceleration towards the negative Z direction. The following options would also be valid : (**1 vector 0.0 0.0 -9.81**) (**-1 vector 0.0 0.0 9.81**).

2.1.2 Exemple 2

Definition of a wall from a '.stl' file

fix **cad1** all mesh/surface **file meshes/Boxbase.stl type 2** scale **0.01**

- The chosen mainWord is of the type 'fix';
- The name of the inserted wall is defined as **'cad1'**;
- The defined wall will interact with all particles of the simulation, thus the targetGroupID is 'all';
- The type of fix being called is 'mesh/surface';
- This type of fix requires the name of the file to be loaded and the type of material (file meshes/Boxbase.stl) it will be made of (type 2).
- In this case, the designed ".stl" file was created in cm. To convert it to the correct unities, it is scaled down by the keyWord keyArgs scale 0.01.

2.2 Script Structure

To execute a simulation, LIGGGHTS requires the definition of several parameters in the correct order. Thus a LIGGGHTS script may be divided into the following sections : Initialization, Execution Details, Model, Execution.:

2.2.1 Initialization

This section describes the simulation environment. It describes the units to be used, boundary behaviors, contact distances, and material parameters. All the following lines must be present in any simulation. #Preliminaries

```
units si
atom_style sphere
atom_modify map array
boundary f f f
newton off
communicate single vel yes
# Declare domain
region domain block -0.15 0.15 -0.4 1.0 -0.10 1 units box
create_box 2 domain
#Neighbor Listing
neighbor $N bin
neigh_modify delay 0
#Material and interaction properties
fix m1 all property/global youngsModulus peratomtype ${Young} ${Young}
fix m2 all property/global poissonsRatio peratomtype ${V} ${V}
fix m3 all property/global coefficientRestitution peratomtypepair 2 ${e} 0.1 0.1 0.1
fix m4 all property/global coefficientFriction peratomtypepair 2 ${u} 0.0 0.0 0.0
fix m5 all property/global coefficientRollingFriction peratomtypepair 2 ${rf} 0.0 0.0 0.0
```

Here is an explanation of the most important lines:

- units : definition of the units for each physical quantity of the simulation;
- atom_style : geometry of the particles simulated;
- boundary : defines how the boundaries of the simulation environment will behave. It is composed of three letters, one for each spatial direction (x,y,z).
  - (p) periodic : particles leaving the simulation through a direction (-x), are reinserted in the opposing direction (+x);
  - (f) fix and non-periodic: any particles leaving the boundary are deleted;
  - (m) variables and non-periodic: boundary increases in size following the particles;

- region : creation of a rectangular region (type 'block') called 'domain' ;
- create_box: definition of the simulation geometrical environment as the region 'domain' and the number N of different materials present (2 in this case).;
- neighbor : distance at which the program search for neighboring particles;
- material properties :
  - The first two commands (youngsModulus et poissonsRatio) defines the parameters for each material. Thus one value is needed per material inserted;
  - The other commands (coefficientRestitution, coefficientFriction et coefficientRollingFriction) define interactions between types of materials. At first, the number of N materials is given (2 in this case) followed by an NxN matrix describing each interaction between pairs of materials;

2.2.2 Détails d'exécution

In this phase, the details of the calculation methods are described.

```
    #Define the particle physics
pair_style gran model hertz tangential history rolling_friction epsd2
pair_coeff * *
#Integrator
fix integrate all nve/sphere
#Gravity
fix grav all gravity 9.81 vector 0.0 0.0 -1.0
#Time step
timestep ${dt}
#Thermodynamic output settings
thermo_style custom step atoms
thermo $T
thermo_modify norm no lost ignore
```

Here is an explanation of the most important lines:

- pair_style : defines the contact model chosen for the interaction between particles. in this case, it is the Hertz model with the addition of rolling friction espd2;
- fix integrate all nve/sphere : choice of which group of particles will have their quantities updated during the simulation. In this case all particles are chosen. All particles that are not being integrated are moved during the simulation.
- timestep : definition of the time passed between two calculations. It is indicated to chose a value smaller than 20% of the Rayleigh time. The latter is defined from the properties of the chosen material as follows:

$$\Delta t_c = \frac{\pi \bar{R}}{0.8766 + 0.163\nu} * \sqrt{\frac{\rho}{G}}$$

- thermo_style : defines which information will be displayed in the command window during the simulation. In this case, the actual step and the number of particles will be displayed.
- thermo: defines the number of timesteps between each information update during a simulation.

2.2.3 Model

During this phase, the simulation execution will be defined. While the previous phases are mostly the same between all simulations, the Model phase will vary depending on the objective of the simulation. Thus, one must describe how grains will be inserted, wall location and movement properties, values saved for post-treatment, etc.

For commands that require a certain level of randomness, a prime number (PN) larger than 10^5 is necessary. For each of these command lines, PN must not be repeated.

- Grain definition To create a particle template **TP**, the material type **T**, the density **D** and the radius **R** must be defined. fix **TP** all particletemplate/sphere **PN** atom_type **T** density constant **D** & radius constant **R**

  Next, the particle distribution will be created. For **NT** particle templates, the template **TP** and the mass fraction **P** of each one must be added to the simulation. It is important to note that the sum of given **P** values should be equal to 1.

  fix **pdd** all particledistribution/discrete **PN NT pts1 P1 pts2 P2** ....

- Insertion

  Different types of insertion are available, in this part the insert/rate/region controlled by the number of particles will be shown. At first, the region where the particles will be created must be defined. In the following a rectangular box is defined by two opposing vertices (x0,y0,z0) and (x1,y1,z1).

  region **factory** prism x0 x1 y0 y1 z0 z1 0 0 0

  Next, the insertion **'ins'** is defined by gravity deposition. The previously defined particle distribution **pdd** will be used. **N** particles will be inserted at a rate of **R** per second between intervals of **E** timesteps. Particles are generated with **constant** speed of **0.0 0.0 -0.1** inside the previously defined region of name **factory**.

  fix **ins** all insert/rate/region seed **PN** distributiontemplate **pdd** nparticles **N** particlerate **R** insert_every **E** vel **constant 0.0 0.0 -0.1** region **factory**

- Wall definition

  Two types of walls can be defined through the usage of an external ".stl" file. Simple walls are either stationary or can be moved at constant speeds using the move/mesh command. Differently, Servo walls can be controlled in stress quantities. The following is an example of the creation of a servo wall :

  fix **servoZ** all mesh/surface/stress/servo file **meshes/PistonZ.stl** type **2** scale **0.01** com **0. 0. 0.** ctrlPV force axis **0. 0. -1.** target_val **TV** vel_max **V** kp **K**

  The geometric description of the wall will be loaded from the file **meshes/PistonZ.stl**. It will be composed of material type 2 and scaled to 0.01 of its original size. The center of mass (com) will not be changed. The controller type will be chosen as force (ctrlPV force), in the direction of -z with a target force value of **TV**. The maximal servo this piston can ever reach is **V** and it uses a proportional controller kp equal to **K**.

  Next, the created mesh must be defined as a wall, and the contact models must be defined in accordance with the pair_style previously chosen.

  fix **walls1** all wall/gran **model hertz tangential rolling_friction epsd2 history mesh** n_meshes **N** meshes **cad1 servoZ ...**

  In this case, the hertz contact model with rolling friction epsd2 was validated for the **N** meshes comprising the model.

- Computes Computes are used to obtain data from the simulation that can later be exported for visualization (".vtk" files) or post treatment (".txt" files). Here are some of the possible computes that can be executed.

  compute ctc all contact/atom #number of contacts between atoms
  compute dpl all displace/atom #particle displacements
  compute pgl all pair/gran/local pos id force contactPoint delta #grain-grain contact
  compute sts all stress/atom #particle stress through Love-Webber
  compute ken all ke/atom #atom kinetic energy
  compute pen all pe/atom #atom potential energy

2.2.4 Exécution

The execution of LIGGGHTS is done through the command "run N", with N being the number of timesteps to be calculated.

3 Avanced Commands

3.1 Variable

There are many types of variables possible for LIGGGHTS script, each with its own utility. All of them are defined through the following line :

variable **ID** type val

In this subsection, the following types of variables will be described: equal, string, index, and loop.

- equal

  This type of variable stores a number (double or integer) that may be used in the continuation of the script. This type of variable is created as following:

  variable **v1** equal 1

  There are two methods to call the value of this variable : ${**v1**} ou v_**v1**

  ex : run ${v1}

  A variable can be created from one equation using previously defined variables or other numbers if the equation. For example, the following script describes how to calculate the average value between **v1** and **v2** :

  ex :variable **v3** equal $((v_**v1**+v_**v1**)/2)

  Another usage of this variable is to account for quantities that are updated during the progress of the simulation. Take for example two opposing pistons P1 and P2 separated by an initial length of **Length**. The distance between the pistons can be calculated by the two following statements :

  ex 1: variable **l1** equal $(v_Length-f_sP1[8]+f_sP2[8])
  ex 2: variable **l2** equal (v_Length-f_sP1[8]+f_sP2[8])

  The only difference between both statements is the presence of a $ before the equation. In LIGGGHTS language, a $ forces a variable to assume a value. Thus, in this example, the variable **l1** will have a fixed value no matter the displacement of the pistons. However, **l2** will be updated for each time step according to the piston displacement.

  A good application of this feature is the control of the stress value required for triaxial tests. As previously mentioned, a servo has a targeted Force value. To define a force value in function of the stress, the area of the piston must be updated with the displacement of the pistons.

- string

  Work just like the variable equal, but stores characters strings instead.

- index

  Index variables store a set of variables that will be used in sequence. It is created as follows:

  variable **v1** index v_V1 v_V2 v_V3 v_V4 v_V5 v_V6 ....

  This kind of variable is used in the same way as the previous ones. When the variable is called, only the active term will be returned. To access the next term of the index variable, the next command must be used.

  next **v1**

  It is impossible to go back to the previous terms of the variable. If a command next is called and there is no other term in the variable, the next jump command in the script will be overridden (not used). This is especially useful for the execution of loops (explained in the next subsection).

- loop

  Loop variables are useful to execute the same command several times as a loop "for" in other programming languages. it is created through the following line.

  variable **v1** loop N

  With N being an integer number, the variable **v1** is equivalent to the variable type index with values going from 1 to N. To access the next value in the **v1** variable, the next is needed.

Many values from LIGGGHTs can be used inside of variables and to export data. To do so, one must know with which kind of mainWord created the valID containing the data.

$$fix \rightarrow f\_valID; \quad variable \rightarrow v\_valID; \quad compute \rightarrow c\_valID;$$

If valID is a vector (or matrix), it is possible to choose one of its values (or entire column) as follows:

$$fix \rightarrow f\_valID[N]; \quad variable \rightarrow v\_valID[N]; \quad compute \rightarrow c\_valID[N];$$

3.2 Jump and Label

Both these commands are used together to create calculation loops, change execution files or jump a section of a script. When a jump command is called, the calculation continues from the targeted label. Take for example the following code :

label **labID**
< - - script - - >
jump filename **labID**

When the jump is called, the program will search for the label **labID** defined in the script "filename" continuing the simulation from there. In this case, the same section script is executed several times. The value "filename" must be defined. Either with the name of the script being executed, or a different script containing the rest of the simulation.

A loop is exemplified in the following :

variable l loop 10
label doStl
< - - script - - >
next l
jump Box.liggghts doStl

In this case, when the variable l reaches N, the next jump value will be overridden, thus leaving the loop.

3.3 Exporting Data

Before talking about exporting data, the different types of data created by LIGGGHTs must be described as each has a unique way of being called. Overall three types can be distinguished **Per-atom**, **global**, and **Local**.

- **Per-atom vectors** Per-atom vectors are either matrices or vectors containing quantities relative to each particle in the simulation as their properties. For each timestep, this quantity is re-updated, thus it does not store quantities over time. To export this kind of data the command dump is used :

  dump ID group-ID style N file args

  Several types of dump exist, but here the types custom and custom/vtk will be explored.

  dump grains all custom $N path/grains*.txt id x y z radius
  This command creates "grains*.txt" files at each interval of N (integer) timesteps, with * representing the timestep the save was executed. The files will contain the id, position, and radius of each grain of the simulation. This command is useful for the exportation of data for post-treatment.

  dump dmpSph all custom/vtk $N path/partcles_*.vtk id x y z radius This command creates "grains*.vtk" files at each interval of N (integer) timesteps, with * representing the timestep the save was executed. The files will contain the id, position, and radius of each grain of the simulation. This command is useful for the visualization of the simulation through the software ParaView.

- **Global vectors** Global vectors contain unique quantities of the simulation that are recalculated over time, for example, the average kinetic energy of the particles or the force being executed by a piston. To export this kind of data the following command is used :

  fix ID group-ID ave/time (average over time).

As an example, the following command is used to store the data related to 2 pistons in the file "path/servoForce.txt". In this case, a new line in the same file is created between N timesteps.

fix servostress all ave/time 1 1 $N f_servoZ[3] f_servoY[2] file path/servoForce.txt

- **Local Vectors**

  Local vectors are values calculated in relation to a region of the simulation. They are data obtained for a certain type of interaction that is not global, for example the contact between grains. To export this kind of data the following command is used :

  dump ID group-ID local

  Take for example the following command to export grains contact data, assuming that compute pgl all pair/gran/local was previously called. At each N timesteps, a new file named "contForce*.txt" will be created containing the data of the 7,8 and 9 column computed by pgl.

  dump contactForces all local $N path/contForce*.txt c_pgl[7] c_pgl[8] c_pgl[10]

3.4 write_data & read_data

The command write_data can be used to save the state and postion of all particles of the simulation. The saved file can then be read by read_data to continue the simulation. This command is specially usefull after grain insertion so several simulations can be run from the same initial state.

write_data filename
read_data filename

3.5 Shell

Shell commands are used to execute command lines as if it was written directly into the terminal. A good usage of this command is to create, delete or displace folders. Another usage is the creation of ".stl" file using a OpenSCAD script and parameters defined into the LIGGGHTS script, as exemplified by the following command :

shell *openscad -Dmode=$stlModes -Dw=$(v_ExpWidth) -Dl=$(v_ExpLength) -Dh=$(v_ExpHeight) -o $stlFname.stl BOX.scad*

In this example, the necessary variables (stlModes, ExpWidth, ExpLength, ExpHeight, stlFname) are exported to the OpenSCAD script "BOX.scad", creating the necessary ".stl" files. This way, if the dimensions of the system are changed, the ".stl" files are regenerated automatically.

3.6 Print

Print command will write the asked information into a log file. This command is useful at the end of a simulation to write down information that may be relevant for the post-treatment. In the following example the log is used to create a new log file to store the initial size of the box for the Matlab post-treatment.

log *LiggghtstoMatlab.txt*
print *"The following values give the Matlab app information about the simulaiton"*
print *""*
print *"ExpWidth=${ExpWidth}"*
print *"ExpLength=${ExpLength}"*
print *"ExpHeight=${ExpHeight}"*

4 Installing LIGGGTHS

Sometimes the installation guide present in the LIGGGHTS documentations may not work for everyone. The following commands should work in all Debian based systems (like Ubuntu or Mint).

- Install LIGGGHTS dependencies : sudo apt-get install openmpi-bin libopenmpi-dev libvtk6.3 libvtk6-dev cmake libavcodec-dev libavformat-dev libavutil-dev libboost-dev libdouble-conversion-dev libeigen3-dev libexpat1-dev libfontconfig-dev libfreetype6-dev libgdal-dev libglew-dev

libhdf5-dev libjpeg-dev libjsoncpp-dev liblz4-dev liblzma-dev libnetcdf-dev libnetcdf-cxx-legacy-dev libogg-dev libpng-dev libpython3-dev libqt5opengl5-dev libqt5x11extras5-dev libsqlite3-dev libswscale-dev libtheora-dev libtiff-dev libxml2-dev libxt-dev qtbase5-dev qttools5-dev zlib1g-dev

- Install git : sudo apt install git
- Clone the source code : git clone https://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git
- open the terminal in the src folder inside LIGGGHTs installation, classic path should be /LIGGGHTS-PUBLIC/src
- command : make auto
- command : gedit MAKE/Makefile.user
- edit the file and make sure the following lines are active : USE_MPI = "ON" ; USE_VTK = "ON" et AUTOINSTALL_VTK = "ON".
- Save and close.
- command : make stubs
- command : make auto
- the file lmp_auto was created. Save it the folder containing liggghts script to be executed.

A video guide can be found in : https://www.youtube.com/watch?v=ru3119ozC6M .

5 OpenSCAD

OpenSCAD is a free and open source software capable of creating the ".stl" files necessary to LIGGGHTs servo functions. This section is a very quick guide to OpenSCAD functionality. More information can be found in the documentation page of the software.

Two principal category of functions will be described : geometries and modifications.

- Geometries
  This functions, as the name indicates, creates base geometries as cubes, cylinders or cones. Each type of geometry has a certain number of parameters that must be entered. The most simple geometry, a cube is created as following :
  cube([X,Y,Z]) - creation of a prism of size (X.Y.Z)
- Modifications
  These types of functions will change the behavior of geometries. Some exemples are translation, rotation, color, intersection and difference.

Examples :

- Red prism with it's reference point in (1,1,0) :

  translate([1,1,0])
  color("red")
  cube([1,1,2,true]);

  The word 'true' when creating a cube makes it's center as the reference point. Without it, it would be one of it's corners.

- Prism with a prismatic whole going through it. with it's reference point in (2,1,3) :

  translate([2,1,3])
  color("blue")
  difference() {
  cube([1.2,1.2, 1], true);
  cube([1,1, 1.2], true);
  };

  The translation and the color are applied to the product of the difference between both objects.

Nomenclature

Physical constants

g	Gravity acceleration	$9,81, m/s^2$
h	Plank constant	$6.62607 \times 10^{-34}\, Js$

Unities

$\bar{R}$	Smallest radius	m
Δt_c	Rayleigh time	s
ν	Poisson Coefficient	–
ρ	Volumetric mass	kg/m^3
E	Young's Modulus	Pa
G	Shearing Modulus	Pa